VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

PHAM VAN PHUC
PHAM DUY KHANH
NGUYEN HUYNH ANH KHOA

MIDTERM REPORT

FLUTTER WITH FIREBASE: AUTHENTICATION, DATABASE, AND CLOUD FUNCTIONS

SOFTWARE ENGINEERING

HO CHI MINH CITY, YEAR 2025

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

PHAM VAN PHUC – 522H0068
PHAM DUY KHANH – 522H0046
NGUYEN HUYNH ANH KHOA – 522H0064

MIDTERM REPORT

FLUTTER WITH FIREBASE: AUTHENTICATION, DATABASE, AND CLOUD FUNCTIONS

SOFTWARE ENGINEERING

Prof., Dr. DUONG HUU PHUOC

HO CHI MINH CITY, YEAR 2025

ACKNOWLEDGMENT

We would like to express our sincere thanks to Mr. Duong Huu Phuoc for his continuous support and enthusiastic guidance throughout the research and completion of our final report.

We would also like to thank the Faculty of Information Technology, Ton Duc Thang University, for providing a rich academic environment. The faculty's willingness to share their expertise and valuable references not only supported our research project but also enhanced our overall learning experience at the university.

At the end of the research project, we look back at the valuable lessons and experiences from our teachers. Although there are still limitations and areas for improvement, we are always willing to learn and develop. We look forward to receiving further guidance to complete our work and appreciate the comments and contributions from our teachers and classmates. With their continuous support, we are determined to improve our research skills in future projects.

We wish all teachers and friends good health and happiness, because everyone's support and care has been an invaluable source of motivation on our journey.

Ho Chi Minh City, day 12 month 04 year 2025

Author

Pham Van Phuc

Pham Duy Khanh

Nguyen Huynh Anh Khoa

This thesis was carried out at Ton Duc Thang University.

Advisor: .....

.....

(Title, full name and signature)

This thesis is defended at the Undergraduate Thesis Examination Committee was hold at Ton Duc Thang University on … /…/……

Confirmation of the Chairman of the Undergraduate Thesis Examination Committee and the Dean of the faculty after receiving the modified thesis (if any).

CHAIRMAN	**DEAN OF FACULTY**
……	**……**

DECLARATION OF AUTHORSHIP

I hereby declare that this thesis was carried out by myself under the guidance and supervision of Mr. Duong Huu Phuoc and that the work and the results contained in it are original and have not been submitted anywhere for any previous purposes. The data and figures presented in this thesis are for analysis, comments, and evaluations from various resources by my own work and have been duly acknowledged in the reference part.

In addition, other comments, reviews and data used by other authors, and organizations have been acknowledged, and explicitly cited.

I will take full responsibility for any fraud detected in my thesis. Ton Duc Thang University is unrelated to any copyright infringement caused on my work (if any).

Ho Chi Minh City, day 12 month 04 year 2025

Author

Pham Van Phuc

Pham Duy Khanh

Nguyen Huynh Anh Khoa

ABSTRACT

This report explores integrating Firebase with Flutter to develop a powerful mobile app, focusing on three key Firebase services: Authentication, Firestore Database, and Cloud Functions. The report provides a comprehensive overview of each component, including their content introduction, implementation, configuration, and real-world use cases. Detailed examples illustrate how to leverage Firebase Authentication for secure user sign-in, Firestore for real-time data management, and Cloud Functions. To demonstrate the practical application of these technologies, a fully functional messaging app was developed, showcasing seamless user authentication, real-time message synchronization, and server-side processing via Cloud Functions. The demo highlights the efficiency, scalability, and ease of integrating Firebase services with Flutter, demonstrating the full functionality of relevant components such as Firebase Authentication, Firebase Database, and Cloud Functions.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Introduction about Flutter

Figure 1 Flutter logo

Flutter is an open source framework developed by Google that helps create cross-platform mobile applications (iOS and Android) with a single codebase. Flutter uses the Dart programming language, supporting the construction of beautiful interfaces, high performance, flexible interface customization and rapid development. Flutter is currently the preferred choice of many developers because of its convenience, speed and good responsiveness on many platforms.

1.2 Introduction about Firebase

Figure 2 Firebase logo

Firebase is a platform developed by Google to support the construction of fast, convenient and highly scalable web and mobile applications. Firebase provides a diverse set of services, helping programmers reduce infrastructure management work, focus on application development, thereby shortening product building time. Firebase can be easily integrated with many different platforms such as Android, iOS, Web, especially with Flutter.

1.3 The Combination of Flutter and Firebase

The combination of Flutter and Firebase creates a powerful and comprehensive solution for modern mobile app development. It is the perfect combination of a powerful frontend framework (Flutter) and a reliable backend platform (Firebase), bringing many outstanding benefits:

- Easy and smooth integration:

  Since both Flutter and Firebase are developed and maintained by Google, they are optimized to work well together. Firebase provides official libraries specifically for Flutter, making the integration process quick and easy.
- Develop quickly with a single codebase:

Flutter allows building cross-platform apps (Android, iOS, Web) from a single source code. Firebase acts as a Backend-as-a-Service (BaaS), helping to manage data, authenticate users, and implement backend logic without having to build a separate server, reducing backend workload, contributing to supporting fast application deployment, shortening development cycles, and helping products get to market quickly.

- High scalability and reliability:

  Firebase has the ability to automatically scale the system according to the number of users and data, helping the application respond well to the rapidly increasing number of users without having to manage the server. Based on Google infrastructure, Firebase ensures high access speed, high uptime, and quick recovery.

- Good support during development:

  Both Flutter and Firebase provide rich official documentation, detailed examples, and a large development community, making the learning and implementation process easier.

1.4 Connect Firebase to Flutter and Initialize Firebase in Flutter

1.4.1 Connect Firebase to Flutter

Steps to connect Firebase to Flutter

1. Initialize Flutter project
2. Set up Firebase Console to create new project
3. Select the platform you want to deploy in Firebase Console
4. Install Firebase CLI

   npm install -g firebase-tools

5. Add required packages in pubspec.yaml file

1.4.2 Initialize Firebase in Flutter

```dart
// Firebase Initialization
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```

Figure 3 Code to initialize Firebase in Flutter

CHAPTER 2: OVERVIEW OF FIREBASE'S MAIN FUNCTIONS

Firebase's core services: Authentication, database (Realtime Database or Firestore), and Cloud Functions.

2.1 Firebase Authentication

A service that provides simple, secure user authentication and management methods

Supports many popular login methods such as:

- Email and password
- Google, Facebook, Twitter
- Phone number (OTP)
- Anonymous authentication

Firebase Authentication helps users register, log in, and manage accounts easily and quickly without the need for programmers to manage a separate authentication infrastructure.

2.2 Firebase Database

Firebase Database includes: Firebase Firestore and Realtime Database. They are both databases but have different organization and operation methods suitable for different purposes.

- Realtime Database Realtime Database
    - Is a NoSQL database, data is synchronized immediately in real time.
    - Suitable for applications that require fast real-time interaction (chat, online games).
    - Data is stored in JSON format, easy to read, write and update.
- Firestore Database
    - Is a newer NoSQL database, providing better scalability than Realtime Database.
    - Stores data in Document - Collection format, making the data structure clearer.
    - Supports more powerful and complex queries than Realtime Database, suitable for large applications that require tight data organization and high scalability.

2.3 Firebase Cloud Function

It is a service that allows you to execute backend code (JavaScript or TypeScript) on Google servers without having to manage the server.

Cloud Functions are triggered by events such as: data updates, new user registration, automatic notifications...

Useful for extending backend features, handling complex logic, sending notifications, automatic payment processing ...

CHAPTER 3: FIREBASE AUTHENTICATION

3.1 Introducing Firebase Authentication

Firebase Authentication is a Firebase service that helps developers build user authentication systems quickly, easily, and securely for mobile or web applications. Firebase Authentication provides strong and secure authentication methods, helping users sign up, log in, and manage accounts with very little code. Firebase Authentication reduces the work of developers when building and maintaining complex user authentication systems, and supports many different authentication methods, easily integrated into Flutter applications.

Firebase Authentication Methods:

- Native providers:

  Native Providers are authentication methods that Firebase supports out of the box, making it easy for users to log in to your app without using an external service.

  - Email/Password: This is the most common and simple authentication method. Users can register and log in with their email and password.
  - Firebase provides an easy-to-use API for user management, including registration, login, password change, and user information management.
  - Phone number: This method allows users to log in using their phone number and an OTP (One-Time Password) sent via SMS. Phone number authentication is often used in areas where email is not common or when users need to log in without using a password.
  - Anonymous: Anonymous authentication allows users to use your app without having to log in or register for an account. This is useful for apps

that don't require users to sign up right away, or when you want to let users try out your app before deciding to create an account.

- Additional providers:

  Firebase Authentication also supports login through popular external services, making it easy for users to log in with their accounts from other platforms.

  - Google: Google Sign-in is one of the most popular methods. Firebase provides a built-in API that makes it quick and secure for users to log in. This method also makes it easy for users to link their Google accounts to Firebase Authentication.
  - Facebook: Users can log in to your app using their Facebook accounts. Firebase has a built-in Facebook Login SDK, making this easy and secure. This method is often used in social apps or games where users can share information from their Facebook accounts.
  - Play Games: This method allows users to sign in with their Google Play Games account. This is a popular method in Android games, helping to synchronize player data and store their game progress.
  - Game Center: For iOS apps, Firebase Authentication supports login via Game Center. This method helps iOS games synchronize player data and manage user profiles easily.
  - Apple: Apple provides a sign-in service using Apple ID accounts on its devices. Firebase supports Apple Sign-In integration on iOS apps, helping users sign in quickly and securely.
  - Github: Users can sign in to your app via their GitHub account, which is useful in software development apps or developer communities.
  - Microsoft: Firebase Authentication also supports sign-in via Microsoft accounts, which is useful for users using Outlook, Office 365, or Microsoft services.
  - Twitter: Twitter Login allows users to log in to your app using their Twitter account. This method is popular in social or information sharing applications.
  - Yahoo: Firebase Authentication also offers a Yahoo login option, which is less popular than other services like Google or Facebook, but is still useful for some users.

- Custom providers

  Firebase also supports custom authentication providers, allowing you to connect your app to authentication services outside of Firebase

  - OpenID Connect: OpenID Connect is an open standard authentication protocol that allows you to use any OpenID provider to authenticate users. This is a flexible method that allows you to connect Firebase Authentication to authentication systems outside of Firebase.
  - SAML: SAML is an XML authentication standard commonly used in large enterprise organizations, allowing you to integrate Firebase Authentication with your existing enterprise authentication system. This method is suitable for organizations that need a centralized user management solution.
  - Advanced: SMS Multi-factor Authentication : SMS Multi-factor Authentication is an advanced security feature in Firebase Authentication. It requires users to not only enter their password but also provide an OTP code sent via SMS to log in. This helps to increase the security of your app, especially when users have access to sensitive data. This is an important feature for apps that require high security, such as banking, payments, or apps that handle important data.

Advantages of Firebase Authentication:

- Easy to integrate: Provides official libraries for Flutter (firebase_auth).
- High security: SSL encryption, complies with popular security standards.
- Multi-platform support: Android, iOS, Web.
- Easy user management: Can track logged in users, lock accounts, require email confirmation, etc.

3.2 Deploy Firebase Authentication

3.2.1 Install Firebase Authentication in Flutter project

In Flutter, you need to install the packages from pub.dev to use Firebase Authentication. The dependencies in the pubspec.yaml and firebase_auth is for Firebase Authentication:

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^3.13.0
  firebase_auth: ^5.5.2
  firebase_storage: ^12.4.5
  cloud_firestore: ^5.6.6
  firebase_database: ^11.3.5
  cloud_functions: ^5.4.0
  google_sign_in: ^6.3.0
```

Figure 4 List of used dependencies

The versions of the dependencies may change depending on the new version of Firebase released.

Import firebase_auth into the .dart files using:

```
import 'package:firebase_auth/firebase_auth.dart';
```

Figure 5 Code import package firebase_auth

3.2.2 Sign up with password account

Description: The user provides an email and password to create a new account.

Implementation:

```
// Sign up with email and password
Future<void> signUpWithEmailPassword(String email, String password) async {
  try {
    await FirebaseAuth.instance.createUserWithEmailAndPassword(
      email: email,
      password: password,
    );
  } catch (e) {
    print('Sign up failed: $e');
  }
}
```

Figure 6 Code to implement sign up with password account

The FirebaseAuth.instance.createUserWithEmailAndPassword() method is used to create a new user account on Firebase Authentication by providing registration information including email and password. When the registration process is successful, this method will return a UserCredential object containing information about the newly created account, including the User and related information such as UID, registered email, etc. Conversely, if an error occurs during the account creation process, the method will throw a corresponding exception. For example, if the email address is already used for another account, the email-already-in-use error will be returned. Similarly, if the provided password is too weak or does not meet Firebase's minimum security requirements, the weak-password error will be triggered.

3.2.3 Sign in with password account

Description: The user logs in with the registered email and password.

Implementation:

```
// Sign in with email and password
Future<void> signInWithEmailPassword(String email, String password) async {
  try {
    await FirebaseAuth.instance.signInWithEmailAndPassword(
      email: email,
      password: password,
    );
  } catch (e) {
    print('Sign in failed: $e');
  }
}
```

Figure 7 Code to implement sign in with password account

The signInWithEmailPassword function is an async function used to perform the user login process with Email and Password via Firebase Authentication. This function takes two parameters, email and password, and then calls the FirebaseAuth.instance.signInWithEmailAndPassword() method to authenticate the login information. If the login process is successful, the user will be authenticated and the login session will be established on Firebase. Conversely, if an error occurs during the login process, such as entering an incorrect password or an email address that does not exist, an exception will be thrown. Common errors include wrong-password, user-not-found, or invalid-email. When an error occurs, a message will be printed to the screen via the print() command. Handling exceptions helps ensure that the application can provide appropriate feedback to the user when the login fails.

3.2.4 Sign up/ Sign in with 3rd party

Firebase Authentication supports other third-party providers like Google Sign-In, Facebook, Twitter, Apple, GitHub, Microsoft, and Yahoo. Each provider requires its own configuration (like App ID for Facebook or Client ID for Apple), but the general process is similar: get an authentication token from the provider and use signInWithCredential to sign in with Firebase.

Google Sign-In was chosen as an example in this report due to its popularity, easy integration with Firebase, and ability to work on both Android and iOS.

Description: Allows users to sign in with their Google account instead of manually entering their Email and Password.

Implementation:

```dart
// Sign up / Sign in with Google
Future<void> signInWithGoogle() async {
  final GoogleSignInAccount? googleUser = await GoogleSignIn().signIn();
  final GoogleSignInAuthentication googleAuth
    = await googleUser!.authentication;

  final credential = GoogleAuthProvider.credential(
    accessToken: googleAuth.accessToken,
    idToken: googleAuth.idToken,
  );

  await FirebaseAuth.instance.signInWithCredential(credential);
}
```

Figure 8 Code to implement sign in with Google

When this function is called, it first displays a dialog that allows the user to select or sign in to their Google account. This is done using the signIn() method of GoogleSignIn, and the result is a GoogleSignInAccount object representing the signed in Google user. Next, the app retrieves the authentication information from the Google account, including the accessToken and idToken, via the authentication property of googleUser. These tokens are used to create a credential using GoogleAuthProvider.credential(). Finally, the app uses these credentials to sign in with Firebase using the signInWithCredential() method of FirebaseAuth.instance. If the authentication is successful, the user is considered to be signed in to the app via Google. The whole process is done asynchronously, ensuring that the UI doesn't hang while waiting for a response from authentication services. This is a popular and convenient way to integrate Google sign-in into modern Flutter apps.

The general process is similar for other providers: obtain an authentication token from the provider, create a credential (e.g., OAuthCredential for Facebook), and call FirebaseAuth.instance.signInWithCredential(). Each provider requires specific configuration in Firebase Console (e.g., App ID for Facebook, Client ID for Apple) and their respective developer platforms. To enable these providers, navigate to Authentication > Sign-in method in Firebase Console and provide the necessary credentials.

3.2.5 Log out

Description: Allows the user to sign out of the current account, disconnecting the session.

Implementation:

```dart
// Logout
Future<void> signOut() async {
  await FirebaseAuth.instance.signOut();
}
```

Figure 9 Code to implement logout

The FirebaseAuth.instance.signOut() method is used to delete the user's current login session in the app. When called, this method will sign the user out of Firebase Authentication, causing any subsequent authentication requests to not be accepted until the user signs in again. For the case of users logging in via Google Sign-In, calling this method will not automatically disconnect the Google account. Therefore, it is necessary to call GoogleSignIn().signOut() to ensure that the Google login session is also closed. Properly handling signout is important in cross-platform apps, especially when using multiple authentication methods.

3.2.6 Reset password

Description: Allows users to recover their password by receiving a password reset email.

Implementation:

```dart
// Reset password
Future<void> resetPassword(String email) async {
  try {
    await FirebaseAuth.instance.sendPasswordResetEmail(email: email);
  } catch (e) {
    print('Password reset email failed: $e');
  }
}
```

Figure 10 Code to implement reset password

The FirebaseAuth.instance.sendPasswordResetEmail() method is used to send a password reset link to the email address provided by the user. When the user receives this email and clicks on the link sent, they will be redirected to a Firebase-provided web page to set up a new password. This is a useful method to assist users in regaining access to their account in case of a forgotten password. The process of sending the password reset email is asynchronous and if the email does not exist or is invalid, an exception will be thrown.

3.2.7 Other additional services

In addition to the main functions presented above, Firebase Authentication also has a number of other outstanding functions.

- Passwordless Authentication:

  Passwordless Authentication, also known as Passwordless Authentication, is an authentication method in which users do not need to enter a password to log in to the application. Instead of asking for a password, the system will send a login link directly to the user's email. After receiving this link, the user only needs to click on it to authenticate and access their account.

  This method increases security because there is no need to store passwords on the system, minimizing the risk of brute-force attacks or password leaks. At the same time, it also improves the user experience by not having to remember passwords. A practical example is logging in with a link from an email when using applications such as Slack or online banking services.

- Two-Factor Authentication:

  Two-factor authentication is an additional security measure to ensure that the user is actually the account owner. After the user logs in using the usual method (e.g. email and password), the system will ask for a second factor for verification. Typically, this second factor is a One-Time Password (OTP) code sent via SMS or email.

  Using 2FA significantly increases security, because even if the user's password is stolen, the attacker still cannot access the account without the OTP code. Common examples are the use of Google Authenticator or OTP codes via SMS when logging into a bank account or cryptocurrency management application.

- Anonymous Authentication:

  Anonymous authentication allows users to access an application without having to create an account or provide login information. When using this method, Firebase creates a temporary account with a unique identifier (UID) for the user.

  This method is often used in applications where the user does not want to register immediately but can still experience some features. For example, in a shopping application, the user can add products to the cart without logging in. Then, if the user decides to register, the temporary data can be associated with the official account.

  However, the limitation of this method is that the anonymous account can be lost when the user deletes the application or changes devices, because there is no way to restore the data without linking to a real account.

- Session Management:

  Session Management in Firebase is handled automatically without complex programming. When the user successfully logs in, Firebase will generate an authentication token and store it on the device (usually in local storage or Secure Storage).

  This token is automatically sent by Firebase with every request to the backend, ensuring that the user is authenticated. Firebase also automatically renews the token when it expires, ensuring that the session remains logged in until the user actively logs out or the session is terminated due to a security error.

  In the event that the user restarts the app, Firebase checks to see if there is a cached token. If so, it automatically authenticates the user without requiring a re-login. This optimizes the experience, especially in apps where users log in frequently.

3.2.8 Manage Firebase Authentication on Firebase Console

Firebase Console is Firebase's official web interface that allows developers to configure, manage, and monitor Firebase Authentication services. From here, you can set up authentication methods, manage users, track login activity, and customize security and user experience settings.

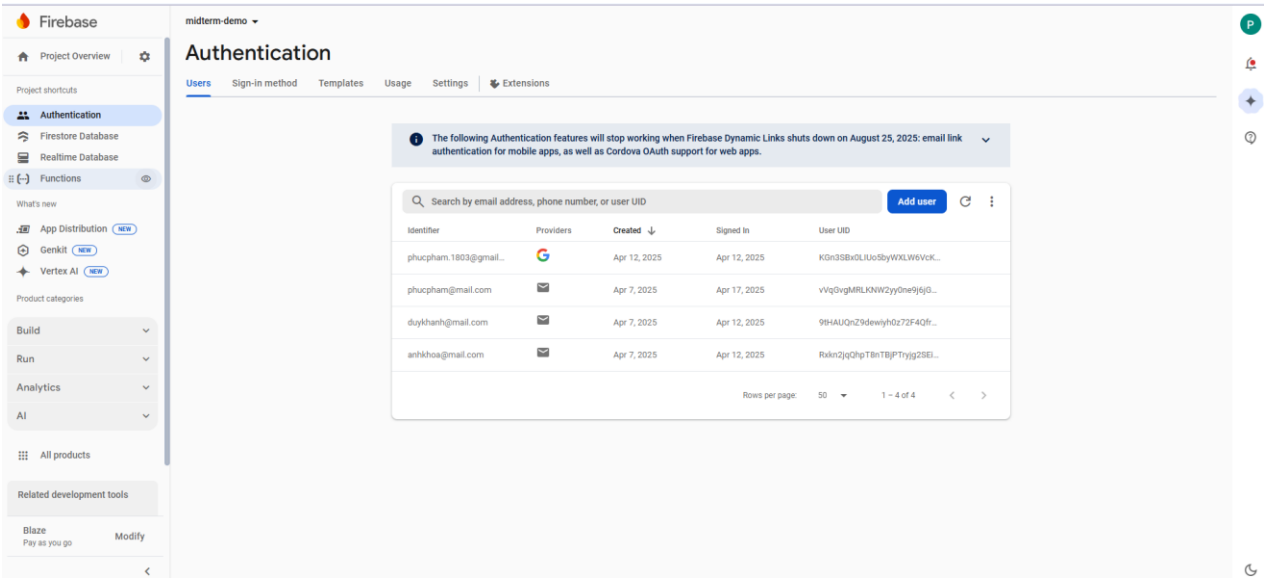

Figure 11 Console screen of Firebase Authentication

Firebase Console provides an intuitive dashboard for managing Firebase Authentication. To access it, log in to Firebase Console, select your project, and navigate to Authentication in the left menu. Here are the key features:

- Configure authentication method:

  In the Sign-in method tab, we can enable or disable the authentication methods that

the application supports, including:

- Email/Password: Allows users to register and log in using email and password.
- Google Sign-In: Enables login using Google account.
- Anonymous: Allows temporary account creation without login information.
- Passwordless (Email Link): Enables login using email link.
- Phone Authentication: Allows authentication via phone number and OTP code.
- OAuth Providers: Supports login using third-party providers such as Facebook, Twitter, GitHub, Apple, ...

- User Management:

  In the Users tab, we can see a list of all users registered in the Firebase Authentication system. Each user is displayed with information such as:

  - UID: Unique user identifier.
  - Email: Associated email address (if applicable).
  - Provider: Authentication method used (Email/Password, Google, Anonymous, etc.).
  - Last Sign-in: Last login time.

  Actions we can take:

  - Delete User: Remove the user account from the system (note: this action cannot be undone).
  - Disable Account: Temporarily lock the user account to prevent login.
  - Update Information: Change the email or reset the password for the user (requires administrative privileges).

  We can also export the user list as CSV for analysis or archiving.

- Track and analyze authentication activity:

  Firebase Console integrates with Firebase Analytics (if enabled) to provide data about login behavior, such as:

  - Number of new users signing up.
  - Variability of logins via methods (Email, Google, etc.).
  - Common authentication errors (e.g. wrong-password, user-not-found).

  This information helps you optimize your authentication process and improve the user experience.

- Customize email template:

  In the Templates tab, you can customize the emails that Firebase sends to users, including:

  - Verification email: Sent when a user needs to verify their email address.
  - Password reset email: Sent when a user requests to recover their password.
  - Passwordless login link email: Sent when using email link authentication.
- Configure security settings:

  In the Settings tab, you can set up security rules, such as:

  - Require email verification: Force users to verify their email before accessing certain features.
  - Limit login attempts: Prevent brute-force attacks by temporarily locking out after multiple failed login attempts.
  - Session expiration: Adjust how long it takes to automatically log out if a user is inactive.

We can also enable Multi-Factor Authentication (MFA) to require two-factor authentication (such as SMS OTP) for accounts.

CHAPTER 4: FIREBASE DATABASE

4.1 Introducing Firebase Database

Firebase offers two popular database services: Realtime Database and Firestore. Both are NoSQL databases, but each has its own unique characteristics that make them suitable for different application needs. Both support data storage and synchronization across devices and platforms, but they differ in scalability, flexibility, and features.

4.1.1 Realtime Database

Firebase Realtime Database is a JSON tree-based NoSQL database that allows for instant data synchronization across all connected devices. It is suitable for applications that require real-time data synchronization, such as chat applications, online games, or location tracking applications.

- Data structure: Data in Realtime Database is stored in JSON, which has a tree-like structure, making it easy to store simple data but sometimes difficult to manage when the data is complex.

  For example:

```
{
  "users": {
    "user1": {
      "name": "Nguyen Van A",
      "age": 30,
      "email": "vana@example.com"
    },
    "user2": {
      "name": "Tran Thi B",
      "age": 25,
      "email": "thib@example.com"
    }
  }
}
```


Figure 12 Example of Realtime Database data structure

- Data synchronization: Realtime Database provides real-time synchronization, which helps data to be updated instantly across all devices and users.
- Speed: The system is optimized to work quickly with applications that need constant data updates and fast synchronization
- Data querying and filtering:

  Firebase provides powerful data query methods, allowing users to search and sort data without loading the entire database.
  - Common methods include:
  - orderByChild(): Sort data by the value of a given child key.
  - orderByKey(): Sort data by key.
  - orderByValue(): Sort data by value.
  - limitToFirst() / limitToLast(): Limit the number of results returned.
  - equalTo(): Filter data by a specific value.
  - startAt(), endAt(): Get data within a range of values.
- Write and update data:

  Firebase supports many methods for writing data to the database including:
  - set(): Overwrite all data at the specified path.
  - update(): Updates a portion of the data without changing other values.
  - push(): Automatically creates a unique ID for each new entry, often used to add new data without overwriting old data.
- Delete Data

  Users can delete data from the Firebase Realtime Database using the remove() method or write a null value to a specific location.

- Security and Data Authentication

  Firebase Realtime Database provides flexible security rules based on the JSON language, allowing control of read and write access to data on a node-by-node basis.

  These rules can be set up in the Firebase Console or as a database.rules.json file.

  Firebase also supports integration with Firebase Authentication to control access on a per-user basis.

  For example: Set users to only be able to read and write data that belongs to them.

```
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "auth ≠ null && auth.uid = $uid",
        ".write": "auth ≠ null && auth.uid = $uid"
      }
    }
  }
}
```

Figure 13 Example of Firebase Authentication rule

- Offline Data Persistence

  Firebase Realtime Database supports offline data storage. When the device loses network connection, the data will be stored locally on the device and automatically synchronized with the server when the connection is restored.

  This is especially useful for applications that need to operate stably in an environment without a continuous network connection (for example, note-taking applications or offline form filling applications).

- Data backup and recovery

  Firebase Realtime Database supports automatic or manual data backup tools from the Firebase Console.

  Data can be exported to JSON format for storage or conversion to other services.

Realtime Database is applied in some cases such as:

- Chat App: Update messages in real time between multiple users. For example: Messenger, Zalo, WhatsApp.
- Realtime Tracker: Track GPS location, device status, or classroom status, IoT devices. Examples: Google Maps, Smart Home Control.
- Voting App: Update voting results in real time. Examples: Online voting app, event survey.

4.1.2 Firebase Firestore

Firebase Firestore is a newer NoSQL database designed to replace the Realtime Database in applications that require scalability and more complex data structures. Firestore stores data as documents in collections, and supports more powerful data queries.

- Data structure: Data in Firestore is organized as documents and collections. A Collection contains multiple Documents, and each Document can contain other Sub-Collections, making it easy to organize complex data and query it. Each document can contain multiple data fields and supports complex data types such as arrays, dates, and nested objects.

  For example:

```
- users (Collection)
  - user1 (Document)
    - name: "Nguyen Van A"
    - age: 25
    - posts (Sub-Collection)
      - post1 (Document)
        - title: "My first post"
        - content: "Hello world"
```

Figure 14 Example of Firebase Firestore data dtructure

- Querying and searching: Firestore provides powerful query capabilities, including complex queries and the ability to sort and filter data based on multiple conditions. This allows you to build applications with complex data query requirements.

  Firestore provides powerful, flexible query methods without loading the entire database. Common methods include:

  - where(): Filter data based on certain conditions.
  - orderBy(): Sort data by a specific field.
  - limit(): Limit the number of results returned.
  - startAt(), endAt(): Get data within a range of values.
  - arrayContains(): Search for elements in an array.

  Query examples:

```
1  FirebaseFirestore.instance.collection('users')
2    .where('age', isGreaterThan: 18)
3    .orderBy('age')
4    .limit(10);
```

Figure 15 Query code example

- Write and update data:

Firestore provides many methods for writing data:

- set(): Completely overwrite data in a Document.
- update(): Update a part of a Document without changing the entire data.
- add(): Add new data to a Collection with an automatically generated ID.

- Delete data

  Firestore supports easy deletion of data:

  - delete(): Delete a Document or Collection.
  - Write null to a specific field to delete that field without deleting the entire Document.

- Scalability: Firestore is designed for better scalability, especially when your app requires processing large amounts of data or has many concurrent users.
- Data Security and Validation

  Firestore provides flexible security rules based on a JSON-like syntax, set up through the Firebase Console or the firestore.rules file.

  Example security rule: only allow users to read and write data that belongs to them

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read, write: if request.auth.uid == userId;
    }
  }
}
```

Figure 16 Example of Firebase Firestore rule

- Offline Data Persistence

  Firestore supports offline data storage. When the device loses network connection, the data is stored locally and will automatically sync to the server when the connection is restored. This is extremely useful for apps that need to operate stably in unstable network conditions.

- Backup and restore data

  Firestore supports manual or automatic data export tools from Firebase Console, making it easy to backup data or convert to other services.

Firebase Firestore is applied in some cases such as:

- User Management App: Store user information: Full name, email, avatar, posts, comments. For example: Blog App, Social Media App (Facebook, Instagram).

- Product Management App (E-commerce App): Store products by category (category is Collection, product is Document). For example: Shopee, Lazada, Amazon.

4.1.3 Compare Firebase Firestore and Realtime Database

Criteria	Firebase Realtime Database	Firebase Firestore
Data structure	Data is stored as JSON trees.	Data is stored as documents and collections.
Query	Simple query, limited filtering and sorting capabilities.	Powerful query support, allowing complex filtering and sorting.
Scalability	Optimized for small and medium-sized applications, limited scalability.	Designed to scale better, ideal for large and complex applications.
Real-time synchronization	Supports instant real-time synchronization between all devices.	Provides real-time synchronization, but not as fast as Realtime Database.
Complex data management	Difficult to manage complex data or relationships between objects.	Manage complex data and relationships between documents more easily.
Security	Basic security rules, easy to set up but not flexible.	More flexible and powerful security rules.
Offline support	Offline support for mobile applications, but limited.	Better offline support and easier offline data management.
Price	Typically cheaper for	Can be more

	applications that require less data synchronization.	expensive for complex query-intensive use cases.
Stability	Widely used, but difficult to scale for large applications.	Designed to be more stable and easy to scale for large-scale applications.

4.1.4 Firebase Storage supports file storage (images, videos)

Although Firebase Storage is not a database, it plays a very important supporting role when combined with Firebase Firestore or Realtime Database to store and manage files such as images, videos, documents. Instead of saving files directly in the database (which is wasteful and slow performance), the app will:

- Upload the file to Firebase Storage.
- Get the download URL from that file.
- Save the URL to Firestore or Realtime Database so that it can be retrieved and displayed again when needed.

4.2 Deploy Firebase Database

4.2.1 Firebase Firestore

4.2.1.1 Install and configure Firebase Firestore for Flutter apps.

In Flutter, you need to install the packages from pub.dev to use Firebase Authentication.

The dependencies in the pubspec.yaml and cloud_firestore is for Firebase Firestore:

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^3.13.0
  firebase_auth: ^5.5.2
  firebase_storage: ^12.4.5
  cloud_firestore: ^5.6.6
  firebase_database: ^11.3.5
  cloud_functions: ^5.4.0
  google_sign_in: ^6.3.0
```

Figure 17 List of used dependencies

The versions of the dependencies may change depending on the new version of Firebase released.

Import cloud_firestore into the .dart files used:

```
import 'package:cloud_firestore/cloud_firestore.dart';
```

Figure 18 Code import package cloud_firestore

4.2.1.2 How to create, read, update, and delete data in Firestore.

- Create Data

  Data will be added to the students collection with an automatic ID.

```
await FirebaseFirestore.instance
  .collection('students')
  .doc('abc123')
  .set({
    'name': 'John Doe',
    'age': 20,
    'email': '
```

Figure 19 Create data code example

- Read Data

  Read data from the document with ID abc123 in the students collection

```
DocumentSnapshot snapshot = await FirebaseFirestore.instance
  .collection('students')
  .doc('abc123')
  .get();
```

Figure 20 Read data code example

- Update Data

  Update only the age field of the document.

```
await FirebaseFirestore.instance
  .collection('students')
  .doc('abc123')
  .update({'age': 23});

```

Figure 21 Update data code example

- Delete Data

  Delete all documents with ID abc123.

```
await FirebaseFirestore.instance
  .collection('students')
  .doc('abc123')
  .delete();
```

Figure 22 Delete data code example

4.2.1.3 Manage Collections and Documents for your app

- Pluralize collection names: users, posts, products, ….
- Use clear Document IDs if you need to do quick operations.
- Avoid nesting too many levels of sub-collections – it affects performance.
- For large data, use pagination (limit, startAfter) to load each part.

4.2.1.4 Perform basic and advanced queries in Firestore.

- Basic Query

  Filter students with age greater than 20

```
QuerySnapshot snapshot = await FirebaseFirestore.instance
  .collection('students')
  .where('age', isGreaterThan: 20)
  .get();

snapshot.docs.forEach((doc) {
  print(doc.data());
});
```

Figure 23 Query basic code example

- Advanced Query

  Get the first 5 students in ascending order of age.

```
QuerySnapshot snapshot = await FirebaseFirestore.instance
  .collection('students')
  .orderBy('age')
  .limit(5)
  .get()
```

Figure 24 Advance query code example

4.2.1.5 Backup and Restore

Firestore doesn't support backing up directly in the Firebase Console. You'll need to use Google Cloud Storage or the gcloud CLI to export/import data. For example:

gcloud firestore export gs://your-bucket/firestore-backup

To enter data, use:

gcloud firestore import gs://your-bucket/firestore-backup

4.2.2 Realtime Database

4.2.1.1 Install and configure Realtime Database for Flutter application.

In Flutter, you need to install the packages from pub.dev to use Firebase Authentication.

The dependencies in the pubspec.yaml and firebase_database is for Firebase Firestore:

```yaml
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^3.13.0
  firebase_auth: ^5.5.2
  firebase_storage: ^12.4.5
  cloud_firestore: ^5.6.6
  firebase_database: ^11.3.5
  cloud_functions: ^5.4.0
  google_sign_in: ^6.3.0
```

Figure 25 List of used dependencies

The versions of the dependencies may change depending on the new released version of Firebase.

Import firebase_database into the .dart files used:

```dart
import 'package:firebase_database/firebase_database.dart';
```

26 Code import package firebase_database

4.2.1.2 Basic operations in Realtime Database (CRUD).

Example structure: /users/{userId}

- Create data

  Create a DatabaseReference object pointing to the path users/123 in the Firebase Realtime Database. Then, use the set() method to write a new object to the database with fields like "name": "John Doe" and "age": 30. If the path users/123 does not exist, it will be created. If it exists, the data there will be completely overwritten.

```dart
DatabaseReference ref = FirebaseDatabase.instance.ref("users/123");
await ref.set({
  "name": "John Doe",
  "age": 30,
});
```

Figure 27 Create data code example

- Read data

  Access the path users/123 and use the get() method to get the data at that location. The result is a DataSnapshot. We check if the data exists using snapshot.exists. If it does, print the value of the data. If not, print the message "No data available."

```
DatabaseReference ref = FirebaseDatabase.instance.ref("users/123");
DataSnapshot snapshot = await ref.get();
if (snapshot.exists) {
  print(snapshot.value);
} else {
  print("No data available.");
}
```

Figure 28 Read data code example

- Update data

  Use update() to update a portion of the data at the path users/123. Specifically, only the "age" field is updated to the new value of 31. The update() method will keep the other data fields and only change the specified field, instead of overwriting the entire field like set().

```
DatabaseReference ref = FirebaseDatabase.instance.ref("users/123");
await ref.update({
  "age": 31,
});
```

Figure 29 Update data code example

- Delete data

  Perform the delete operation at the path users/123. By calling remove(), all user information at this location will be deleted from the database. After this operation, users/123 will no longer exist in the Realtime Database.

```
DatabaseReference ref = FirebaseDatabase.instance.ref("users/123");
await ref.remove();
```

Figure 30 Delete data code example

4.2.1.3 Query data in Realtime Database

Query all records in the users path where the age value is 30. First, we call orderByChild("age") to sort the records by the age field value, then combine it with equalTo(30) to filter out the matching records. Then, we call once() to execute the query once and iterate through the returned records using a for loop.

```dart
DatabaseReference ref = FirebaseDatabase.instance.ref("users");
Query query = ref.orderByChild("age").equalTo(30);

DatabaseEvent event = await query.once();
for (final child in event.snapshot.children) {
  print(child.value);
}
```

Figure 31 Query data code example

4.2.1.4 Listen to real-time data (Realtime Listeners)

Set up a listener using onValue.listen() to monitor any data changes in the users node. Whenever the child data inside users changes (add, delete or update), the callback function will be called and print out all the latest data at that time. This is a realtime mechanism unique to Firebase that helps the application always synchronize data with the server.

```dart
DatabaseReference ref = FirebaseDatabase.instance.ref("users");

ref.onValue.listen((DatabaseEvent event) {
  final data = event.snapshot.value;
  print("Realtime update: $data");
});
```

Figure 32 Realtime Listeners code example

4.3 Manage Firestore and Realtime Database on Firebase Console

4.3.1 Firestore

Firebase Console provides an intuitive interface for managing Firestore data, setting security rules, and monitoring activity.

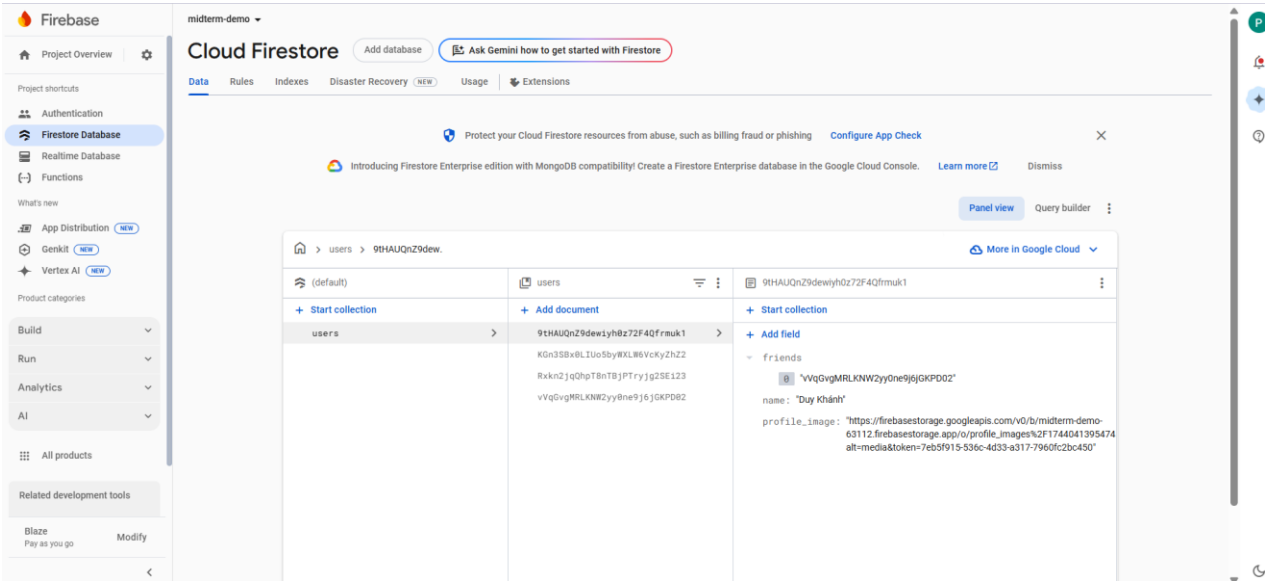

Figure 33 Console Screen of Firestore

- Manage data: In Firestore Database > Data, you can:
  - View data: Click a collection or document to view its JSON content.

- Add data: Click Add collection or Add document, enter the ID and fields with values (string, number, boolean, etc.).
- Edit data: Click a document, edit a field, or add a new field.
- Delete data: Select a document/collection and click the trash can icon to delete it.
- Filter data: Use the Filter list button to filter documents by field or value.
- Manage paths: Use the Edit path button to access a specific collection/document (e.g. /users/user123).

- Manage security rules:
  - In the Rules tab, you can write and test Firestore Security Rules to control access.
  - Use Rules Simulator to test rules with simulated scenarios (read, write, delete) without affecting real data.
  - After editing, click Publish to apply the rule.
- Manage Indexes:

  In the Indexes tab, you can:
  - Create Indexes: Support complex queries (like WHERE with ORDER BY).
  - Remove Indexes: Remove unnecessary indexes to optimize costs.

  If the query fails due to missing indexes, Firebase Console provides links to create the necessary indexes.
- Monitor Usage:
  - In the Usage tab, view statistics on reads, writes, deletes, and costs over time (day, week, month).
  - Integrate with Cloud Monitoring to receive alerts when usage thresholds are exceeded.

4.3.2 Realtime Database

Firebase Console allows you to manage data, security rules, and backups.

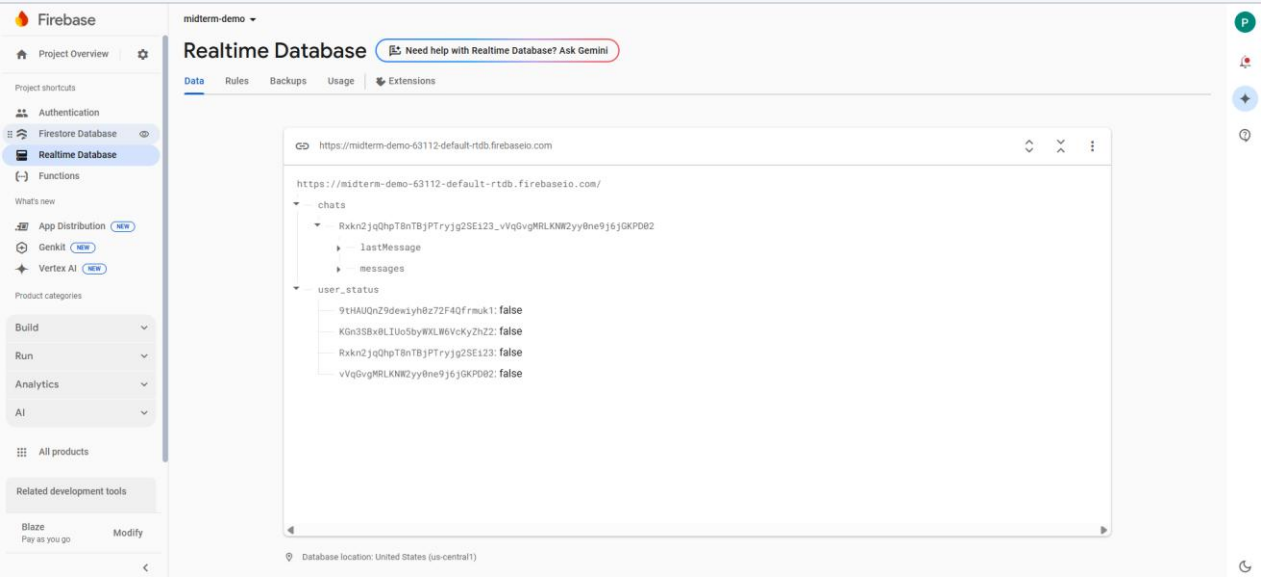

Figure 34 Console Screen of Realtime Database

- Manage Data:

  In Realtime Database > Data, you can:

  - View data: Data is displayed as a JSON tree, with nodes and values.
  - Add data: Click + next to a node to add a child node or value (string, number, object).
  - Edit data: Click on a value to edit it directly.
  - Delete data: Select a node and click the trash can icon.
- Import/export data:
  - Export JSON: Select the root node, click Export JSON to download a JSON file.
  - Import JSON: Click Import JSON and select a file to restore the data.
- Manage Security Rules: In the Rules tab, write Realtime Database Security Rules to control access
- Monitor usage:
  - In the Usage tab, view statistics on connections, data traffic, and storage.
  - Use the Rules Simulator to test rules with simulated scenarios.
  - Click Publish to apply the rule.
- Usage monitoring:
  - In the Usage tab, view statistics on connections, data traffic, and storage.
  - Check out the Realtime Database Usage Dashboard for more detailed analysis.
- Back up your data:

- In the Backups tab, enable automatic backups or perform manual backups.
- Download a backup file from Google Cloud Storage or restore using:

CHAPTER 5: FIREBASE CLOUD FUNCTION

5.1 Introducing Firebase Cloud Function

Firebase Cloud Functions is a Firebase service that lets you write and deploy backend code without having to manage servers. Cloud Functions operates as a serverless service, where you can focus on writing logic without having to worry about deploying or maintaining infrastructure. This backend code is triggered by events from Firebase or HTTP APIs, and can perform tasks like processing data, sending notifications, or interacting with other services in Firebase.

Firebase Cloud Functions is part of Firebase Functions, and it is a server-side coding solution that lets you write functions to handle events without having to manage servers. These functions can be triggered automatically when events occur in your app. These events can include:

- Changes to data in Firestore or Realtime Database
- Authentication of new users (sign up or sign in)
- Errors in the app or system events
- Events that send notifications or interact with external APIs

Cloud Functions help handle server-side logic without having to build and maintain your own servers. Instead, you just write functions and Firebase automatically manages requests and scales as needed. This simplifies app deployment and maintenance.
Key uses of Firebase Cloud Functions include:

- Automatically handle events: Events from Firebase, such as changes to data in Firestore, Realtime Database, or user authentication requests, can automatically trigger Cloud Functions that you write. For example, you can write a function to automatically send an email or notification to a user when they successfully sign up.
- Interact with external services: Firebase Cloud Functions can be used to integrate with external services like payments, email, or third-party APIs. You can send HTTP requests to these services and receive responses right in your Cloud Function.

- Manage and handle complex logic: Cloud Functions can help you handle complex server-side logic without having to deploy and maintain separate servers. For example, handling payments or handling tasks like calculating data in your app.

Firebase Cloud Functions offers many benefits, especially when you don't want to manage servers or need to handle server-side tasks without worrying about infrastructure. Here are some reasons why you should use Cloud Functions in your app:

- Handle logic without servers:

  One of the most important reasons to use Firebase Cloud Functions is that you don't have to manage servers. You don’t need to build or maintain complex backend servers, nor do you need to worry about managing infrastructure, security, or scalability. Cloud Functions automatically scales to meet your needs and only charges for the resources you use, saving you money.
- Automate event handling:

  Cloud Functions makes it easy to automate processes in your app without having to build complex management systems. For example, when a user signs up for an account or changes data in Firebase, you can write functions to automatically send emails, notifications, or update data elsewhere without having to call them manually.
- Support complex tasks:

  Cloud Functions lets you perform complex server-side tasks like processing payments, interacting with external APIs, or processing data based on specific conditions. This increases the flexibility and scalability of your app without having to rely on complex backend systems.
- Automatic Scaling:

  Firebase Cloud Functions is built on Google Cloud's serverless platform, so it can scale automatically without your intervention. As your app grows and requires more resources, Cloud Functions will automatically scale and handle requests efficiently without running into scalability issues.
- Easy Integration with Other Firebase Services

  Cloud Functions is optimized to integrate with other Firebase services like Firestore, Realtime Database, Firebase Authentication, Firebase Cloud Messaging,

and more. This makes it easy for you to handle events and interact with these services without having to write a lot of code.

In Firebase, Cloud Functions are divided into different types depending on how they are triggered, serving specific needs in the application.

- One of the most common types is HTTP-triggered functions, which are triggered when an HTTP request is sent. This type is often used to build backend APIs or handle requests from the client side such as saving contact forms, sending data, or performing custom processing.
- Next is Callable functions, which are a special type of HTTP functions, allowing clients (like Flutter or web apps) to directly call functions from the application using httpsCallable. The advantage of callable functions is the ability to automatically check user authentication and the simplicity of passing parameters and receiving responses.
- In addition, Firebase also supports triggers from Realtime Database and Firestore, which means that when there is a change in data (such as creating, updating, or deleting), a function will be automatically called to handle the next logic. For example, when a user sends a message, you can trigger a Cloud Function to check the content, filtering out sensitive words before saving it to the database.
- There are also Authentication-triggered functions, which are used to handle when a new user registers or is removed from the system. These are often used to send a welcome email or initialize the user's profile data. Finally, Storage-triggered functions are triggered when there is a change to a file in Cloud Storage, such as generating a thumbnail when a user uploads a photo.

Examples of tasks that can be done with Firebase Cloud Functions:

- Send a welcome email when a user registers: When a new user registers in your app, you can write a Cloud Function to automatically send a welcome email or confirm the user's account.
- Processing payments: You can use Cloud Functions to process payments through external services, like Stripe or PayPal, and update order status in Firebase.
- Updating data after changes: Cloud Functions can be triggered when data in Firestore or Realtime Database changes, allowing you to update or sync data with other systems.

- Sending notifications: You can use Firebase Cloud Messaging (FCM) to send notifications to users when there is a new event, such as a new message or status update.
- Handling user-related events: For example, when a user logs in, you can write a Cloud Function to store the user's information in Firestore or log their login.

5.2 Deploy Firebase Cloud Function

5.2.1 Install and configure Firebase Cloud Functions for Flutter apps.

In Flutter, you need to install the packages from pub.dev to use Firebase Cloud Functions.

The dependencies in the pubspec.yaml and cloud_function is for Firebase Cloud Functions:

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^3.13.0
  firebase_auth: ^5.5.2
  firebase_storage: ^12.4.5
  cloud_firestore: ^5.6.6
  firebase_database: ^11.3.5
  cloud_functions: ^5.4.0
  google_sign_in: ^6.3.0
```

Figure 35 List of used dependencies

The versions of the dependencies may change depending on the new version of Firebase released.

Import cloud_function into the .dart files using:

```
import 'package:cloud_functions/cloud_functions.dart';
```

Figure 36 Code import package cloud_function

5.2.2 Deploy Cloud Functions:

Firebase Cloud Functions is a serverless backend service that allows developers to write server-side logic in languages like JavaScript or TypeScript. When developing a Flutter app, integrating Cloud Functions helps separate the logic from the client, increasing the security and scalability of the system.

Use the firebase init functions command to initialize the functions/ folder containing all the source code of the Cloud Functions. At this step, the developer can

choose the language to write the function: JavaScript (more accessible) or TypeScript (supports more explicit and secure data types). After the folder structure is created, writing the processing functions will be done inside the index.js file (or index.ts if using TypeScript).

A Cloud Function can be triggered by many types of events such as writing/reading from the Realtime Database, user registration, or calling directly from the Flutter app using the httpsCallable method. After completing the function writing part, to deploy them to Firebase, we use the command firebase deploy --only functions. This command will compile the source code, package and upload it to Google's cloud system. The deployed functions will appear in the Firebase Console and can be monitored for logs, call frequency, and processing performance through the intuitive management interface.

5.2.3 How to call and trigger Cloud Functions from Flutter apps.

Once installed, calling a Cloud Function is done by creating a callable function with a name that matches the name declared in the index.js file on the server. For example, with a function named filterAndSendMessage.

```dart
Future<void> sendFilteredMessage(String chatId, String message) async {
  final callable = FirebaseFunctions.instance.httpsCallable('filterAndSendMessage');

  try {
    final result = await callable.call({
      'chatId': chatId,
      'senderId': FirebaseAuth.instance.currentUser?.uid,
      'content': message,
    });

    print('Send succesfully: ${result.data}');
  } catch (e) {
    print('Error when calling Cloud Function: $e');
  }
}
```

Figure 37 Example code of Cloud Function

The call() function will send input data to the Cloud Function and receive a response from the server.

5.2.4 Manage and monitor Cloud Functions

Firebase provides a variety of tools to help developers easily monitor the performance of Cloud Functions after they have been deployed.

After a successful deployment, the entire list of Cloud Functions will be displayed in the Functions section of the Firebase Console.

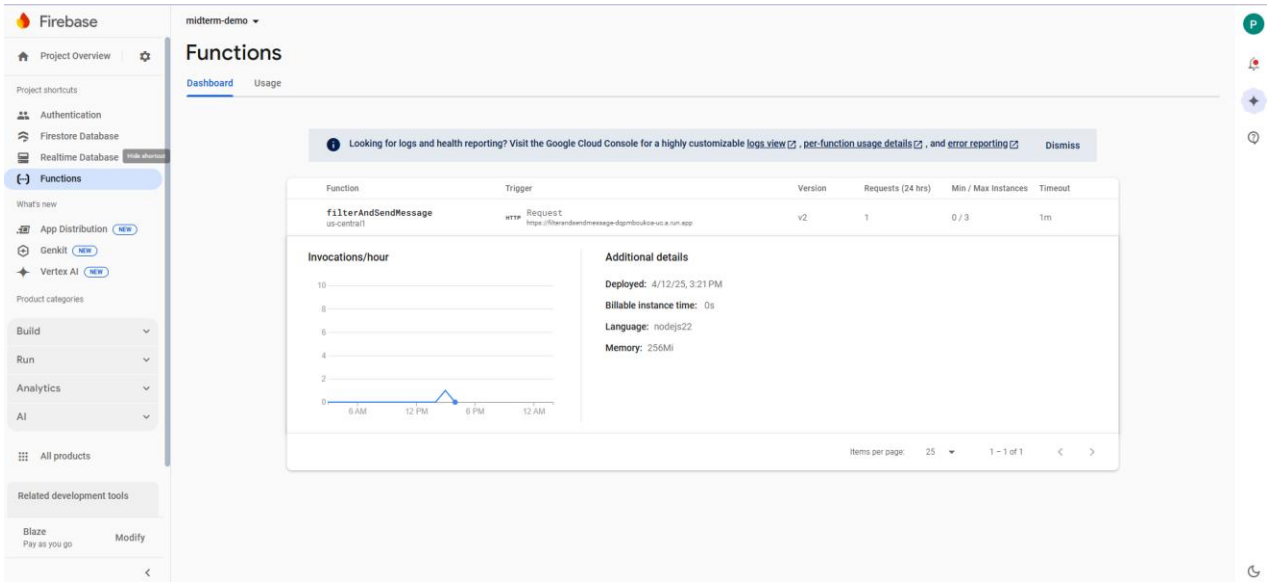

Figure 38 Console Screen of Cloud Function

Users can:

- List of deployed functions: function name, last update time, last activation.
- Performance statistics: number of calls, average processing time, error rate, amount of data exported/imported.
- Error and warning history: shows the times the function failed during execution, with detailed error messages.

In addition to the web interface, Firebase also supports Firebase CLI to monitor functions via the command line. For example:

- firebase functions:log – View logs of Cloud Function calls
- firebase functions:log --only yourFunctionName – View logs for a specific function
- firebase deploy --only functions – Re-deploy functions when there are changes

CHAPTER 6: DEMO APPLICATION

6.1 Demo application description

The demo app is a real-time messaging platform simulation app, built with Flutter and integrating Firebase core services. The app allows users to register, log in, connect with friends, send and receive messages, edit sent messages, delete messages, and update activity status (online/offline) in real time.

In terms of user authentication, the app has fully implemented Firebase Authentication, allowing users to register and log in with email and password, as well as log in with Google account. User information after authentication is saved and synchronized to the system to serve chat and friend connection related features.

The system uses Cloud Firestore to store user data and friend list. Main functions such as creating, editing, and deleting data have been fully implemented. In addition, the Realtime Database's real-time synchronization feature helps to instantly display changes on the user interface when data is updated, such as when there is a new message, the user changes the status, or edits/deletes a message. In addition, the application has also successfully deployed Cloud Functions to serve automated tasks such as handling events when there is a new message, sending push notifications, or logging important changes in the system. Using Cloud Functions helps increase the scalability and automation of the system, while reducing the processing logic on the client side.

6.2 Functions performed corresponding to Firebase services

6.2.1 Firebase Authentication

- Sign up with password account
  Users can create an account with email and password. The registration process will be authenticated through Firebase Authentication .
- Sign in with password account
  Users log in with the previously registered email/password. After successful authentication, the application will go to the main screen with full personal information and friends/chat list.
- Sign up/login with Google

6.2.2 Firebase Database

6.2.2.1 Firebase Firestore

Once a user signs up (via email or Google), information like name, email, profile picture, friends list, etc. will be saved to Firestore. This makes it easy to manage user profiles and sync across multiple devices.

6.2.2.2 Realtime Database

- Display user activity status: Every time a user is online or exits the application, the "online/offline" status will be updated immediately to the Realtime Database. The friend list will be listened to in real time to display a green dot when the friend is active.
- Messaging: The system supports messaging between two users, helping them easily exchange information in real time. All messages are stored in the Firebase Realtime

Database in the structure chats/{chatId}/messages. Each message includes the content, sender, timestamp, and is arranged in chronological order to ensure the conversation is displayed in the correct order. The chat interface is designed to update automatically when there is a new message without having to manually reload the page. This gives users a smooth experience, similar to popular messaging applications today.

- Additionally, users can edit the content of a sent message, in case of needing to correct spelling errors or update information. When a message is edited, the system will update the content in the Realtime Database.
- In addition, users can also delete messages they have sent. Deleted messages will be removed from the database. Editing and deleting messages are both done in real time, helping the conversation reflect the current status between the parties.

6.2.3 Firebase Cloud Function

Before saving the message to the database, the message content is sent to a Cloud Function (filterAndSendMessage). This function will check and filter inappropriate words, replace them with ***, and then save them to the Realtime Database.

CHAPTER 7: CONCLUSION

Through the process of research and practical implementation, it can be seen that Flutter combined with Firebase provides a comprehensive, powerful and modern solution for developing cross-platform mobile applications. Firebase provides all the necessary services for the backend such as user authentication (Authentication), real-time data storage and synchronization (Realtime Database, Firestore), as well as complex server-side processing through Cloud Functions.

The integration of Firebase Authentication makes registration and login faster, supporting multiple authentication methods such as email, password, and Google Sign-In, thereby improving the user experience. Realtime Database allows updating and displaying activity status, messages in real time - suitable for messaging applications. Firestore is more flexible and suitable for storing structured information such as user profiles, friend lists, etc. Finally, Cloud Functions helps handle complex and sensitive logic such as filtering message content securely on the server side, enhancing security and efficiency.

Overall, the combination of Flutter and Firebase provides an efficient mobile application development platform, saving development time while ensuring strong features, security and good extensibility.

REFERENCES

1. FlutterFire - https://firebase.flutter.dev/
2. Firebase Document - https://firebase.google.com/docs?hl=en